**Embedded System Software CSE4116**

**HW#3 Project :**

**Stopwatch module with**

**module programming + interrupt programming**

**컴퓨터공학과**

**20131579 윤기영**

1. **개발 목표**

Module programming, 디바이스 드라이버 구현, interrupt 등 실습 시간 때 배운 내용을 활용하여 간단한 stopwatch 프로그램을 작성한다

1. **개발 내용 및 필요 배경지식**

* **fpga\_fnd device driver, timer module, interrupt를 포함한 stopwatch 기능을 가진 module을 구현**

fpga\_fnd의 초기값은 [0000]이고 fpga\_fnd부분의 앞의 두 자리는 분(60분), 뒤의 두 자리는 초(60초)를 표시한다. 입력은 board의 Home, Back, VOL+, VOL- 버튼으로 받으며 각 버튼이 눌려졌을 때 interrupt가 발생한다.

**Home 버튼 :**

스탑워치가 작동한다. 해당 버튼은 초기 상태에서만 작동한다. kernel timer를 사용하여 1초마다 fnd의 시간을 update해준다.

**Back 버튼 :**

작동 중이던 스탑워치를 일시 정지시킨다. 이 때 소수점 이하의 시간을 유지한다. 예를 들어 1.9초에 일시정지를 하고 다시 재개하면 0.1초만에 시간이 갱신된다. 일시정지를 해제하려면 Back 버튼을 다시 누른다.

**VOL+ 버튼 :**

스탑워치를 리셋시킨다. 정확히는 시간만 리셋시킨다. 스탑워치가 작동 중일 경우에는 fnd의 시간이 0000으로 되고 1초마다 fnd가 갱신된다. 스탑워치가 일시정지 상태일 경우, fnd의 시간만 0000으로 바뀐다.

**VOL- 버튼 :**

3초 이상 누르고 있을 시, user application을 종료시키고 fnd의 값을 0000으로 초기화한다. user application은 이 버튼을 3초 이상 누르고 있지 않는 한, 종료되지 않는다.

* **stopwatch 기능을 가진 module을 실행시키는 application을 구현**

스탑워치 device를 open하고 write를 호출하여 device driver 내에서 sleep하여 wait\_queue에 들어간다. 만약 board의 VOL-버튼을 3초 이상 누르고 있을 시, sleep에서 깨어나 device를 close하고 프로그램이 종료된다.

* **interrupt**

일반적으로 하드웨어의 통신 속도는 프로세서 속도보다 훨씬 느리기 때문에 하드웨어가 응답할 때까지 커널이 요청을 보내고 기다리는 방식은 이상적이지 않다. Polling은 커널이 주기적으로 시스템 하드웨어의 상태를 확인하고 그 상태에 따라 처리하는 것이다. 하지만 확인 작업을 불필요하게 반복하므로 시스템에 불필요한 부하를 준다. 그래서 커널 처리가 필요한 순간에 하드웨어가 커널에게 신호를 보낼 수 있는 체계를 만들었는데 이것이 interrupt이다. interrupt는 프로세스 클럭과 상관없이 비동기적으로 발생되는 것이 특징이다.

* **interrupt handler**

interrupt를 처리하기 위해 커널이 실행하는 함수를 interrupt handler 또는 interrupt service routine이라고 부른다. interrupt handler는 device driver에 들어있다. interrupt handler가 일반적인 커널 함수와 다른 점은 interrupt가 발생했을 때 커널이 호출한다는 점과 interrupt context라는 특별한 context에서 실행된다는 점이다. 이 context에 있는 코드를 실행하는 동안에는 실행을 중단할 수 없다. interrupt는 언제든지 발생할 수 있으며 interrupt가 발생하면 중단된 코드를 최대한 빨리 다시 실행하기 위해 interrupt handler의 실행 속도가 빨라야 한다. 그런데 interrupt handler가 방대한 양의 연산이나 많은 메모리 연산을 처리해야 할 수도 있으므로 실행 속도가 빨라야 한다는 목표달성이 어려울 수 있다. 이를 해결하기 위해서 interrupt를 받은 즉시 실행되며 처리시한이 중요한 작업만을 처리하는 top half, 나중에 할 수 있는 일은 bottom half에서 처리하도록 분리시켜 놓았다.

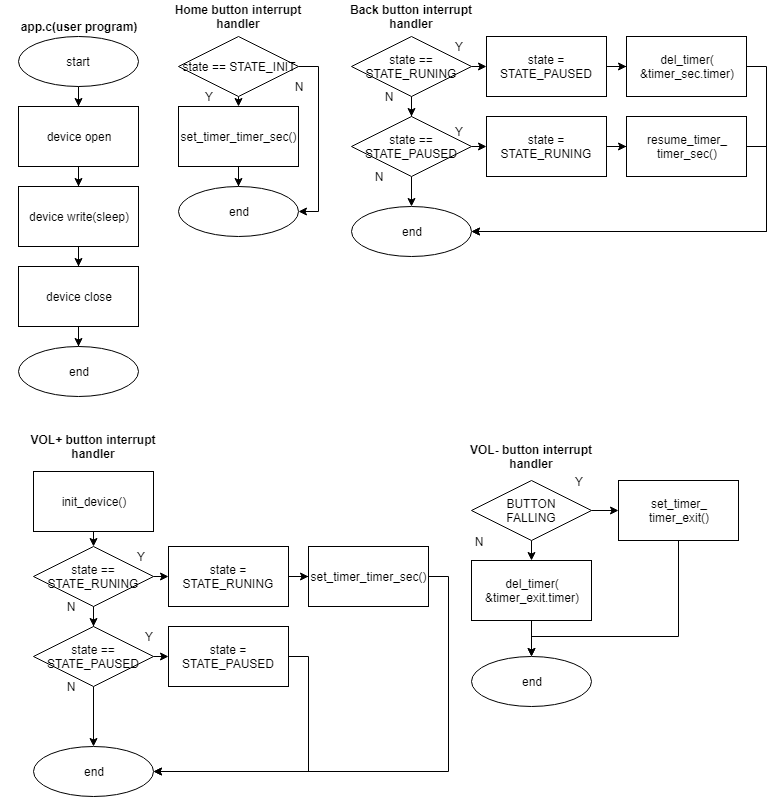
interrupt handler는 “int request\_irq(unsigned int irq, irq\_handler\_t handler, unsigned long flags, const char \*name, void \*dev)”함수를 사용하여 등록한다. 각 인자는 interrupt 번호, 실제 interrupt handler를 가리키는 함수 포인터, interrupt 상태를 지정하는 플래그, 장비의 이름, interrupt 공유 시 사용하는 변수를 의미한다.

1. **개발 일정**

|  |  |
| --- | --- |
| 날짜 | 개발 내용 |
| 2019.05.23 | - stopwatch.c(device driver) code 작성  - interrupt handler 작성  - device driver Makefile 작성 |
| 2019.05.24 | - comment 추가 및 stopwatch.c 작성  - stopwatch 종료 기능 구현  - app.c(user program) 작성 |
| 2019.05.27 | - kernel warning 관련된 부분 모두 수정  (del\_timer\_sync(), gpio 관련) |
| 2019.05.28 | - 보고서 작성 |

1. **개발 결과**

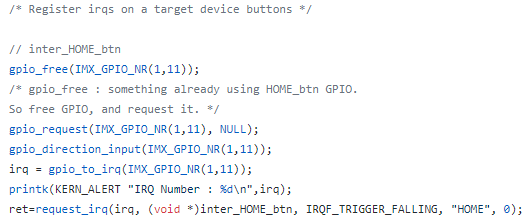
* **Flow chart**



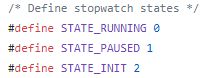
* **fpga\_fnd device driver, timer module, interrupt를 포함한 stopwatch 기능을 가진 module을 구현**

스탑워치 기능을 가진 모듈인 stopwatch.c를 작성하였다. 먼저 insmod하면 실행되는 inter\_init()함수에서는 fnd device를 mmap시키고 이후에 시간을 잴 때 사용할 timer\_sec와 timer\_exit의 timer변수를 init\_timer()함수를 통해 초기화해주었다.

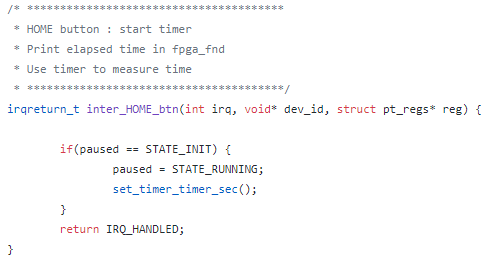
device를 open하면 device에서 사용하는 여러 변수들을 초기화해주고, 각 버튼에 해당하는 pin들을 할당하고 interrupt handler를 등록한다. 종료 버튼인 VOL-버튼을 제외하고는 모두 버튼이 눌릴 때 interrupt가 발생하도록 하였다.



스탑워치는 STATE\_RUNNING, STATE\_PAUSED, STATE\_INIT, 총 3개의 state를 가진다. 각각 작동 중, 일시 정지 중, 초기 상태를 의미한다.



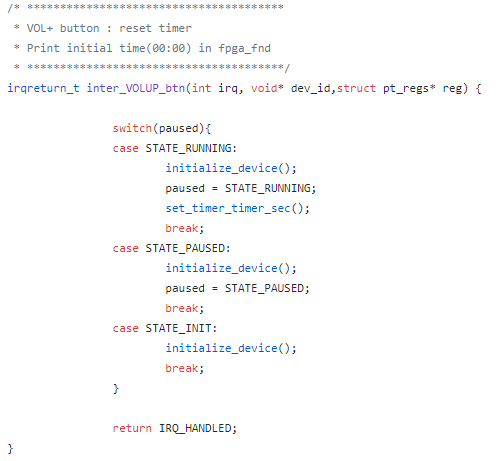
초기 상태에서는 Home 버튼을 통한 start 기능만 수행할 수 있다. Home 버튼을 누르면 state가 STATE\_RUNNING이 되며, 1초마다 expire function이 호출되는 timer\_sec가 set된다.



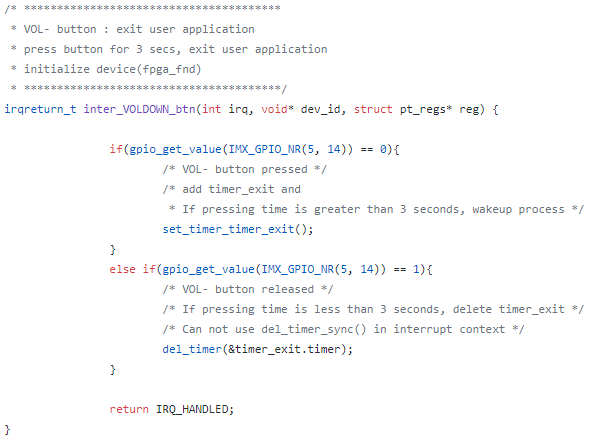
스탑워치가 작동 중에 일시 정지 기능을 하는 Back 버튼을 누르면 state가 STATE\_PAUSED가 되며, 마지막 timer interrupt가 걸릴 때의 jiffies값과 현재 jiffies값의 차이를 계산하여 소수점 이하의 시간 값을 저장해 둔다. 그리고 timer\_sec의 timer를 제거한다. 만약 다시 Back 버튼이 눌리면 일시 정지가 풀리게 된다. state는 STATE\_RUNNIGN으로 다시 바뀌며, 아까 저장해 두었던 jiffies값을 사용하여 일시 정지 이전에 흐른 시간도 계산하여 1초를 다시 카운트한다.



리셋 기능을 하는 VOL+버튼이 눌리면, fnd의 값을 모두 0000으로 초기화 시킨다. 만약 스탑워치가 작동 중인 상태여도 fnd의 값만 0000으로 바뀌며, 그 상태에서 1초가 지나면 카운트가 계속 증가한다. 해당 코드에서는 initialize\_device()에서 fnd값을 update해준다.



VOL-버튼을 3초 이상 누르면 종료하도록 구현하기 위해서 interrupt handler 내부에서 버튼이 눌렸을 때 발생한 interrupt인지, 버튼을 떼었을 때 발생한 interrupt인지 구분할 수 있게 gpio\_get\_value()를 사용하였다. 만약 버튼을 눌렀을 때 방생한 interrupt라면 3초짜리 타이머를 세팅하는 set\_timer\_timer\_exit()함수를 호출한다. 해당 함수에서는 3초가 지나면 잠들었던 프로세스를 wake up시키므로 user program은 종료하게 된다. 만약 여기서 3초가 지나기전에 버튼을 뗀다면 interrupt가 발생할 텐데, 이 때는 아까 버튼을 눌렀을 때 설정해둔 timer\_exit타이머를 삭제시킨다. 그러면 결과적으로 3초 동안 버튼을 눌러야만 유저 프로그램이 종료되도록 구현이 된다.



interrupt context에서는 del\_timer\_sync()를 사용하면 kernel warning이 뜬다. 이 모듈은 SMP환경에서 돌아가지 않는다는 가정이므로 del\_timer()함수를 사용해서 kernel warning을 없앴다.

또한 device open시, gpio\_direction\_input()함수 호출 시 kernel warning이 뜨는데, 이는 버튼에 해당하는 pin이 사용 중이라 등록이 되어있는 듯하여 gpio\_free()를 통해 먼저 할당을 해제한 후에 등록해주었다.

* **stopwatch 기능을 가진 module을 실행시키는 application을 구현**

스탑워치 디바이스를 사용하는 user program인 app.c를 작성하였다. 디바이스를 open하고 해당 디바이스가 제대로 open되었는지 확인 후, write()함수를 호출하였다. 해당 모듈에서는 write할 경우 process를 sleep시킨다. process는 계속 sleep된 상태로 board의 스위치를 통해 스탑워치가 조작된다. 만약 종료하는 버튼을 3초 이상 눌러 process가 wakeup하게 되면 user program은 디바이스를 close한 뒤 종료한다.

* **기능**

fpga\_fnd의 초기값은 [0000]이고 fpga\_fnd부분의 앞의 두 자리는 분(60분), 뒤의 두 자리는 초(60초)를 표시한다.

* **Home 버튼 :**

스탑워치가 작동한다. 해당 버튼은 초기 상태에서만 작동한다. kernel timer를 사용하여 1초마다 fnd의 시간을 update해준다.

* **Back 버튼 :**

작동 중이던 스탑워치를 일시 정지시킨다. 이 때 소수점 이하의 시간을 유지한다. 예를 들어 1.9초에 일시정지를 하고 다시 재개하면 0.1초만에 시간이 갱신된다. 일시정지를 해제하려면 Back 버튼을 다시 누른다.

* **VOL+ 버튼 :**

스탑워치를 리셋시킨다. 정확히는 시간만 리셋시킨다. 스탑워치가 작동 중일 경우에는 fnd의 시간이 0000으로 되고 1초마다 fnd가 갱신된다. 스탑워치가 일시정지 상태일 경우, fnd의 시간만 0000으로 바뀐다.

* **VOL- 버튼 :**

3초 이상 누르고 있을 시, user application을 종료시키고 fnd의 값을 0000으로 초기화한다. user application은 이 버튼을 3초 이상 누르고 있지 않는 한, 종료되지 않는다.

1. **기타**

* **소감**

인터럽트를 직접 사용해보면서 인터럽트와 인터럽트 핸들러에 대한 이해가 더 깊어졌다. 다만 정확히 인터럽트가 일어나는 플로우에 대해서는 빈 퍼즐이 많아서 더 공부를 해보아야 할 필요를 느낀다. 또한 전반부 처리, 후반부 처리에 대한 내용 역시 이번 프로젝트에서는 다루지 못하기도 해서 따로 공부를 해야 할 것 같다. 전반적으로 kernel warning에 대한 것은 모두 구글링을 통해 해결해서 나름 깔끔하게 짠 듯한 느낌이다.